

KOMPILÁTORY: Syntaktická analýza (Metódy zdola-nahor 1)

Jana Dvořáková

`dvorakova@dcs.fmph.uniba.sk`



Úlohy syntaktickej analýzy

- 1 Číta postupnosť tokenov generovanú lexikálnym analyzátorom a konštruuje strom odvodu/syntaktický strom
 - Overuje, či vstupný reťazec môže byť generovaný CF gramatikou pre vstupný jazyk
 - 2 Hlásí syntaktické chyby "inteligentným" spôsobom
 - Vie sa zotaviť z bežných chýb tak, aby sa mohlo pokračovať v spracovávaní vstupu
- Pracuje na úrovni *bezkontextových jazykov*

Metódy syntaktickej analýzy

① Univerzálne metódy

- Cocke-Younger-Kasami, Earley
- Všetky CF gramatiky, ale neefektívne ($O(n^3)$)

② Metódy zhora-nadol ✓

- Konštruujú strom odvodenia od koreňa k listom v preorderi (ľavé krajné odvodenie)
- Rekurzívny zostup (prediktívne parsovanie)
- LL gramatiky
- Aj ručne implementované parsery

③ Metódy zdola-nahor *

- Konštruujú strom odvodenia od listov ku koreňu (pravé krajné odvodenie)
- Operátorovo-precedenčné parsovanie
- LR gramatiky
- Zvyčajne parsery skonštruované automatickými nástrojmi

Parsovanie zdola-nahor

- Pre vstupný reťazec sa konštruuje strom odvodenia od listov ku koreňu
- Hovoríme o *redukcii* vstupného reťazca na počiatočný symbol gramatiky
 - V každom redukčnom kroku je podreťazec zodpovedajúci pravej strane nejakého pravidla nahradený neterminálom z ľavej strany
- Cieľom je získať pravé krajné odvodenie
- Používa sa metóda posun-redukcia (shift-reduce)

Niektoré pojmy

- Rukoväť (handle)
 - Podreťazec, ktorý zodpovedá pravej strane nejakého pravidla a ktorého redukcia predstavuje jeden krok pravého krajného odvodu
 - Formálne:
 - Handle vetnej formy γ je pravidlo $A \rightarrow \beta$ a pozícia v γ , kde sa β môže nachádzať a ak ju nahradíme neterminálom A získame predchádzajúcu pravú vetnú formu
 - Postupná redukcia:
$$W = \gamma_n \leftarrow_{rm} \gamma_{n-1} \leftarrow_{rm} \dots \leftarrow_{rm} \gamma_1 \leftarrow_{rm} \gamma_0 = S$$
 - 1 Lokalizácia handle β_n v γ_n
 - 2 Nahradenie β_n ľavou stranou pravidla $A \rightarrow \beta_n$
 - 3 Opakujeme pre $n - 1, n - 2, \dots$ až kým nedostaneme S
- Životaschopný prefix
 - Prefix pravej vetnej formy, ktorý nepresahuje pravý koniec (najpravejšej) handle vetnej formy

Metóda posun-redukcia (shift-reduce)

- Všeobecná metóda parsovania zdola-nahor
- Pri implementácii sa využíva zásobník a vstupný buffer
- Akcie parsera:
 - 1 *shift* (posun): vstupné symboly sa ukladajú do zásobníka
 - 2 *reduce* (redukcia): keď sa na vrchu zásobníka objaví nejaká handle, redukuje sa podľa príslušného pravidla gramatiky
 - 3 *accept*: úspešné ukončenie parsovania
 - 4 *error*: volanie funkcie na ošetrovanie chyby
- Handle sa vždy objaví na vrchu zásobníka, nikdy nie vo vnútri!
- Získame pravé krajné odvodenie zapísané odzadu

Metóda posun-redukcia (shift-reduce)

Konflikty

- Nedeterministické rozhodnutia:

1 Lokalizácia handle → *shift/reduce* konflikt

```
Pr.  stmt  →  if expr then stmt
          |  if expr then stmt else stmt
          |  other
```

2 Výber pravidla pre redukciu → *reduce/reduce* konflikt

```
Pr.      stmt  →  id ( parameter_list )      / * procedure call * /
          stmt  →  expr := expr
          parameter_list → parameter_list , parameter
          parameter_list → parameter
          parameter  →  id
          expr  →  id ( expr_list )          / * array reference * /
          expr  →  id
          expr_list →  expr_list , expr
          expr_list →  expr
```

Metóda posun-redukcia (shift-reduce)

Deterministické riešenia

- Deterministický shift-reduce parser sa musí vedieť jednoznačne rozhodnúť pre akciu na základe
 - 1 obsahu zásobníka
 - 2 k symbolov zo vstupu
- $LR(k)$ gramatiky
 - Umožňujú deterministické spracovanie, nespôsobujú konflikty
 - V kompilátoroch sa zvyčajne využívajú $LR(1)$ gramatiky s výhľadom jeden symbol
- 1 Operátorovo-precedenčné parsovanie
- 2 LR parsovanie

Operátorovo-precedenčné parsovanie

- LR gramatiky
 - Najväčšia trieda gramatík, pre ktorú vieme implementovať deterministický shift-reduce parser
- Operátorovo-precedenčné gramatiky
 - Podmnožina LR gramatík, umožňuje *jednoduchú* implementáciu shift-reduce parsera
 - Okrem iného:
 - Pravá strana pravidla nemôže byť ε
 - Neterminály na pravej strane pravidla sa nesmú vyskytovať priamo vedľa seba
 - Vhodné pre parsovanie aritmetických výrazov

Precedenčné relácie

- Definujeme tri disjunktné precedenčné relácie:

$a \succ b$ a "má väčšiu precedenciu ako" b

$a \doteq b$ a "má rovnakú precedenciu ako" b

$a \prec b$ a "má menšiu precedenciu ako" b

- Relácia medzi dvojicou terminálov sa určuje:

① Intuitívne

② Zostrojením jednoznačnej gramatiky a použitím mechanickej metódy na odvodenie precedenčných relácií

- Precedenčné relácie slúžia na ohraničenie handle

- \prec začiatok handle
- \doteq stred handle
- \succ koniec handle

Použitie precedenčných relácií

- Pravá vetná forma oper.-prec. gramatiky je tvaru:

$\beta_0 a_1 \beta_1 \dots a_n \beta_n$, kde β_i je ε alebo neterminál a a_i je terminál

- Vo vetnej forme ignorujeme neterminály a medzi každé dva terminály vložíme príslušnú precedenčnú reláciu
 - Reťazec je ohraničený symbolom \$ a pre všetky terminály b platí $\$ < b$ a $b > \$$
- Lokalizácia handle:
 - 1 Prechádzaj vstupným reťazcom, kým nenájdeš prvé $>$.
 - 2 Vráť sa naspäť (doľava) cez všetky \doteq , kým nanájdeš $<$.
 - 3 Handle obsahuje všetko medzi nájdeným $<$ a $>$, vrátane vnútorných a okolitých neterminálov
- Neterminály neovplyvňujú parsovanie, nie je potrebné medzi nimi rozlišovať
- Implementácia pomocou zásobníka

Použitie precedenčných relácií

Príklad

- Gramatika $G = (N, T, P, S)$, $N = \{E\}$, $T = \{id, +, *\}$,
 $E \rightarrow E + E \mid E * E \mid id$

- Precedenčná tabuľka:

	id	$+$	$*$	$\$$
id		$>$	$>$	$>$
$+$	$<$		$<$	$>$
$*$	$<$	$>$		$>$
$\$$	$<$	$<$	$<$	

- Spracovanie reťazca $id + id * id$:

$\$ < id > + < id > * < id > \$$

$\$ < + < id > * < id > \$$

$\$ < + < * < id > \$$

$\$ < + < * > \$$

$\$ < + > \$$

$\$ \$$

Parsovací algoritmus

nastav ip na prvý symbol $w\$$;

repeat forever

if $\$$ je na vrchu zásobníka a ip ukazuje na $\$$ **then**

return

else begin

 nech a je symbol navrchu zásobníka

 a b je symbol, na ktorý ukazuje ip ;

if $a < b$ alebo $a \doteq b$ **then begin**

 vloží b do zásobníka;

 posuň ip na ďalší vstupný symbol

end

else if $a > b$ **then**

repeat

 vyber symbol zo zásobníka;

until terminál navrchu zásobníka je vo vzťahu $<$

 k naposledy vybranému terminálu

else error()

end

Určenie precedenčných relácií

- Binárne operátory:
 - Ak operátor θ_1 má väčšiu precedenciu ako operátor θ_2 potom $\theta_1 \succ \theta_2$ a $\theta_2 \prec \theta_1$.
 - Ak majú operátory θ_1, θ_2 rovnakú precedenciu (môže to byť aj ten istý operátor) potom
 - Ak sú ľavoasociatívne tak $\theta_1 \succ \theta_2$ a $\theta_2 \succ \theta_1$
 - Ak sú pravoasociatívne tak $\theta_1 \prec \theta_2$ a $\theta_2 \prec \theta_1$
 - Pre všetky operátory θ :
 - $\theta \prec id, id \succ \theta$
 - $\theta \prec (, (\prec \theta$
 - $\theta \succ),) \succ \theta$
 - $\theta \succ $, $ \prec \theta$
- Unárne operátory (ν je unárny prefixový operátor):
 - $\theta \prec \nu$ pre ľubovoľný operátor θ (unárny aj binárny)
 - $\nu \succ \theta$ ak ν má väčšiu precedenciu ako θ ,
 $\nu \prec \theta$ ak ν má menšiu precedenciu ako θ
- Problémom sú operátory, ktoré majú viac precedencií (mínus)

Kompresia precedenčnej tabuľky

- Precedenčná tabuľka sa v praxi kóduje dvomi precedenčnými funkciami f a g , ktoré zobrazujú terminály na celé čísla
- Pokúšame sa ich definovať tak, aby platilo:
 - 1 $f(a) < g(b)$ ak $a < b$
 - 2 $f(a) = g(b)$ ak $a \doteq b$
 - 3 $f(a) > g(b)$ ak $a > b$
- Relácia medzi a a b môže byť určená numerickým porovnaním $f(a)$ a $g(b)$
- Stratia sa chybové záznamy
 - Zväčša sa to nepokladá za veľkú nevýhodu, lebo chyby môžu byť ešte detekované pri volaní redukcie (nájde sa handle, ale nezodpovedá mu pravá strana žiadneho pravidla)

Konstruktoria precedenčných funkcií

Vstup: Precedenčná tabuľka

Výstup: Precedenčné funkcie pre danú precedenčnú tabuľku, alebo odpoveď, že neexistujú

- 1 Vytvor symboly f_a a g_a pre každý terminál a a \$.
- 2 Rozdeľ vytvorené symboly do čo najviac skupín takým spôsobom, že ak platí $a \doteq b$ potom f_a a g_b sú v tej istej skupine.
- 3 Vytvor orientovaný graf, ktorého uzlami sú skupiny vytvorené v bode 2. Pre každé a a b
 - Ak $a < b$ pridaj hranu $g_b \rightarrow f_a$
 - Ak $a > b$ pridaj hranu $f_a \rightarrow g_b$
- 4 Ak výsledný graf obsahuje cyklus, precedenčné funkcie neexistujú. Ak neobsahuje cyklus, nech
 - $f(a)$ je dĺžka najdlhšej cesty začínajúcej v skupine obsahujúcej f_a
 - $g(a)$ je dĺžka najdlhšej cesty začínajúcej v skupine obsahujúcej g_a

Ošetrenie chýb

- Dva druhy chýb:
 - 1 Nájde sa neredukovateľná handle (neexistuje pre ňu v gramatike pravidlo)
 - "Zlá"handle sa vyberie zo zásobníka
 - Namiesto redukcie sa vypíše diagnostická správa - určí sa porovnávaním handle s pravými stranami pravidiel
 - 2 Prázdne miesto v parsovacej tabuľke
 - Vykoná sa lokálna korekcia na vrchu zásobníka alebo vo vstupnom bufferi (vloženie znaku, zmena znaku, zmazanie znaku)
 - Zotavenie nesmie spôsobiť zacyklenie

Operátorovo precedenčné parsovanie

Zhrnutie

- Výhody
 - Jednoduchá implementácia efektívneho parsera
- Nevýhody
 - Ťažko sa narába s tokenmi, ktoré majú viac precedencií (napr. mínus - môže byť unárne aj binárne)
 - Slabá previazanosť parsera s gramatikou - niekedy si nie sme istí, že parser akceptuje práve zdrojový jazyk
 - Metódu je možné použiť iba pre malú množinu gramatík
- Často sa používa pre parsovanie výrazov a konštrukcie na vyššej úrovni sa rozpoznávajú rekurzívnym zostupom

LR parsovanie

- Efektívna technika parsovania zdola nahor
- Môže byť použitá pre veľkú podmnožinu bezkontextových gramatík - LR(k) gramatiky
 - L - vstupný reťazec sa číta zľava doprava
 - R - vytvára sa pravé krajné odvodenie
 - (k) - výhľad dopredu
- Tri techniky konštrukcie LR parsera
 - 1 Jednoduchý LR parser (simple LR - SLR)
 - Najjednoduchší na implementáciu, ale najslabší
 - 2 Kanonický LR parser (LR)
 - Najsilnejší, ale najnáročnejší na implementáciu
 - 3 LR parser s výhľadom (lookahead LR - LALR)
 - Kompromis medzi SLR a LR

LR parovanie

Výhody a nevýhody

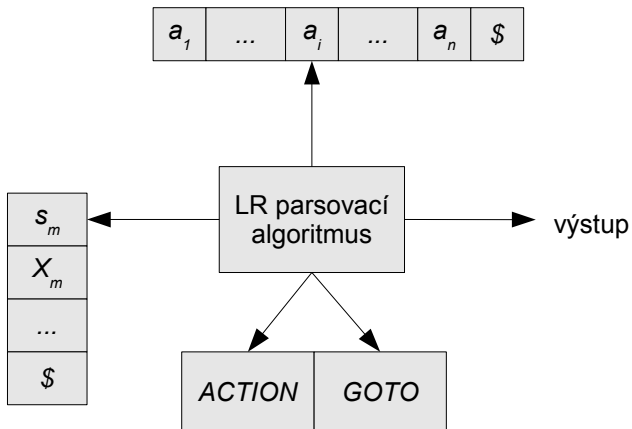
(+) Výhody:

- Umožňuje rozpoznávať takmer všetky programovacie jazyky
- Je to najsilnejšia metóda bez backtrackingu pre shift-reduce parovanie
- LR gramatiky sú vlastnou nadmnožinou LL gramatík
- Chyby sa detekujú najskôr, ako je to možné

(-) Nevýhody:

- Náročná implementácia
- Zvyčajne sa využívajú automatické generátory LR parserov (Bison)

Model LR parsera



Parsovacia tabuľka

- Funkcia $action[s_m, a_i]$:
 - Riadi parsovanie
 - Argumentami sú aktuálny stav a práve čítaný symbol vstupu
 - Vráti akciu
 - ① shift s
 - ② reduce $A \rightarrow \beta$
 - ③ accept
 - ④ error
- Funkcia $goto[s, a]$:
 - Pomocná funkcia, volá sa vždy po redukcii
 - Argumentami sú stav a symbol gramatiky, vráti nový stav
- Shift akcie a goto tabuľka tvoria prechodovú funkcia DKA rozpoznávajúceho životaschopné prefixy

Konfigurácie LR parsera

- Dvojica (*obsah zásobníka, neprečítaný vstup*)

$$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \$)$$

- Reprezentuje pravú vetnú formu: $(X_1 X_2 \dots X_m a_i a_{i+1} \dots a_n)$
- Parser vykoná akciu podľa $action[s_m, a_i]$, konfigurácie sa menia nasledovne:

- 1 shift s :

$$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m a_i s, a_{i+1} \dots a_n \$)$$

- 2 reduce $A \rightarrow \beta$:

$$(s_0 X_1 s_1 X_2 s_2 \dots X_{m-r} s_{m-r} A s, a_i a_{i+1} \dots a_n \$),$$

kde $s = goto[s_{m-r}, A]$ a $r = |\beta|$

- 3 accept: ukončenie parsovania
- 4 error: volanie funkcie na ošetrovanie chyby

LR parsovací algoritmus

nastav ip na prvý symbol $w\$$;

repeat forever begin

 nech s je stav na vrchu zásobníka a

a je symbol, na ktorý ukazuje ip ;

if $action[s, a] = \text{shift } s'$ **then begin**

 vlož a a potom s' na vrch zásobníka;

 posuň ip na ďalší vstupný symbol

end;

else if $action[s, a] = \text{reduce } A \rightarrow \beta$ **then begin**

 vyber $2 * |\beta|$ symbolov zo zásobníka;

 nech s' je stav navrchu zásobníka;

 vlož A a potom $goto[s', A]$ na vrch zásobníka;

 vypíš na výstup pravidlo $A \rightarrow \beta$

end

else if $action[s, a] = \text{accept}$ **then**

return

else error()

end

LR gramatiky

- Vieme pre ne zostrojiť LR parsovaciu tabuľku
- Existujú CF gramatiky, ktoré nie sú LR
 - Vo všeobecnosti sa im vieme vyhnúť pri popise syntaxe programovacích jazykov
- Handle sa rozpoznáva podľa
 - 1 Obsahu zásobníka
 - Nečítame v každom kroku celý zásobník. informáciu o jeho obsahu reprezentuje stav DKA na vrchu zásobníka
 - 2 k vstupných symbolov
 - V praxi nás zaujímajú prípady $k = 0$, $k = 1$

LR vs. LL gramatiky

- LR:
 - Pravidlo sa rozpoznáva podľa všetkého, čo sa odvodí z jeho pravej strany a k symbolov výhľadu dopredu
- LL:
 - Pravidlo sa rozpoznáva podľa neterminálu na ľavej strane, a k symbolov z toho, čo sa odvodí z jeho pravej strany